Conrad sur INTERNET
www.conrad.fr

N O T I C E

Version 07/11

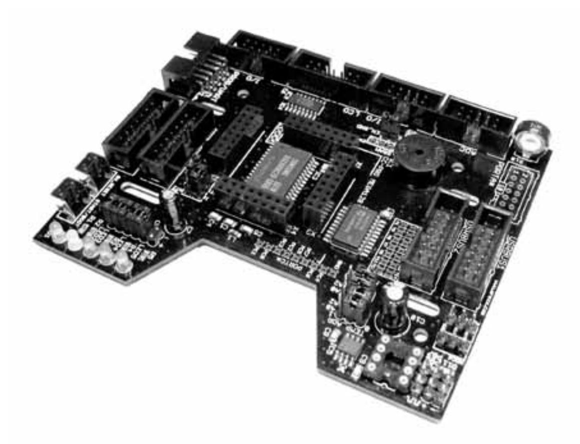
Module extension RP6 CC-PRO MEGA128

Code : 191563

Cette notice fait partie du produit. Elle contient des informations importantes concernant son utilisation. Tenez-en compte, même si vous transmettez le produit à un tiers.

Conservez cette notice pour tout report ultérieur !

RP6 C-Control PRO M128
Module d'extension



⚠IMPORTANT
A lire absolument

Avant de mettre en marche votre module d'extension RP6, il est important de lire entièrement ce mode d'emploi, celui du SYSTEME ROBOT RP6 et celui du C-Control PRO ! Ils contiennent des explications importantes quant à une utilisation correcte et vous informent d'éventuels dangers possibles ! De plus, ils possèdent des informations importantes, qui ne sont nullement évidentes pour la plupart des utilisateurs. Le mode d'emploi du RP6 CCPRO M128 n'est qu'un complément !

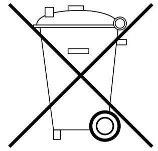
En cas de non observation de ce mode d'emploi et de celui du système robot RP6, toute garantie prend fin ! De plus, AREXX Engineering ne sera tenu pour responsable pour tout type de dommages, résultant d'une non observation de ce mode d'emploi !

Prêtez toute votre attention au paragraphe « consignes de sécurité » du mode d'emploi du SYSTEME ROBOT RP6 !

Remarques sur la garantie limitée et la responsabilité

La garantie de AREXX Engineering se limite à l'échange ou à la réparation du robot et de ses accessoires, dans le délai de garantie légale en cas de fautes de production avérées, comme des dommages mécaniques, des pièces d'équipement manquantes ou fausses, exceptés tous les composants branchés à des connecteurs/socles.

B. Recyclage et consignes de sécurité



Recyclage

Le RP6 et les composants annexes ne doivent pas être éliminés dans les ordures ménagères ! Le RP6 et ses accessoires doivent être apportés à une déchetterie à tri sélectif ou bien à une collecte d'appareils électriques usagés, comme tout appareil électroménager !

Si vous avez des questions à ce propos, renseignez-vous auprès de votre commerçant.

Consignes de sécurité pour les accus et les piles

Les accus et les piles ne sont pas à laisser entre les mains des enfants ! Ne laissez pas les piles/accus traîner à la vue de tout le monde, les enfants ou les animaux domestiques risqueraient de les ingérer. En cas d'ingestion, consulter immédiatement un médecin !

Les piles qui fuient ou abîmées peuvent provoquer des brûlures en cas de contact avec la peau, dans ce cas, utilisez des gants de protections adaptés ! Veillez à ce que les piles/accus ne soient pas mis en court-circuit ou ne soient pas jetés dans le feu. Les piles normales ne doivent pas être rechargées ! Il existe un risque d'explosion ! Seuls les accumulateurs conçus pour être rechargés, comme par exemple les accus NiMH, peuvent être chargés avec un chargeur adapté !

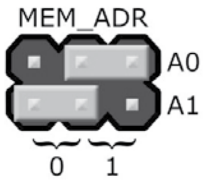
Recyclage des accus et des piles

Tout comme le robot, les accus et les piles ne doivent pas être éliminés dans les ordures ménagères ! En tant qu'utilisateur final, vous êtes tenus par la loi de restituer toutes les piles et tous les accus usagés ! Il est strictement interdit de les jeter dans les ordures ménagères !

De ce fait, amenez vos accus usagés/vieux ou vos piles vides à votre commerçant ou à une collecte de piles de votre commune ! Vous pouvez également amener les accus et les piles usagés dans tous les lieux ou des accus et des piles sont vendus.

Vous remplissez ainsi vos obligations légales et contribuez à la protection de l'environnement.

Adresse EEPROM bus I²C :



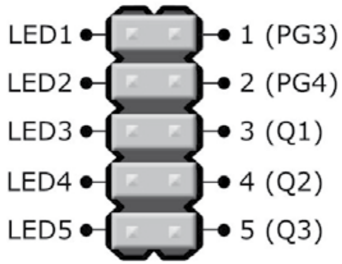
Egalement deux bits (A0 et A1) de l'adresse d'un EEPROM optionnel à emboîtement peuvent être modifiés.

Tenez compte de la fiche technique de l'EEPROM ! Le schéma d'adressage peut varier en fonction de chaque constructeur !

L'adresse de l'EEPROM correspond en écriture binaire à : 10100[A1][A0]0

Les adresses possibles sont : 0xA6, 0xA4, 0xA2 et 0xA0 (160 décimales)

LED :



Les LED se désactivent par jumper. Dans le champ de ce jumper, vous pouvez utiliser, entre autres, des I/O ou des sorties de commutation directement sur le connecteur. LED1 et 2 sont connectées avec les ports I/O PG3 et PG4, LED3, 4 et 5 seulement à des sorties propres au registre à glissement, auquel est aussi branché le LCD.

Il n'existe aucune responsabilité pour des dommages résultant directement ou après l'utilisation du robot. Les réclamations reposant sur des consignes légales nécessaires sur la responsabilité du fait des produits restent intactes.

Dès que vous procédez à des modifications irréversibles sur le robot ou accessoire (par exemple : soudage de pièces supplémentaires, forage d'autres trous, etc.) ou que le robot subit des dommages suite à la non observation de ce mode d'emploi, toute garantie prend fin !

Il ne peut être garanti, que le logiciel fourni avec chaque version suffit ou qu'il est possible de travailler entièrement avec sans interruption ou erreur.

De plus, le logiciel fourni est modulable et peut être chargé dans l'appareil par l'utilisateur. Ainsi, l'utilisateur prend un risque quant à la qualité et l'efficacité de l'appareil y compris tous les logiciels.

AREXX Engineering garantit la fonctionnalité de l'exemple d'application fourni tout en observant les conditions spécifiées dans les données techniques. S'il est prouvé que le robot ou le logiciel pc est en plus défectueux ou insuffisamment, le client prend en charge tous les coûts pour le service, la réparation ou la correction.

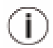
Tenez également compte de l'accord de licence du CD-ROM !

Symboles

Les différents symboles suivants sont utilisés dans ce mode d'emploi :



Le symbole « attention » indique un paragraphe particulièrement important, que vous devez soigneusement respecter. Si vous effectuez une erreur, cela peut conduire à une destruction du robot ou de ses accessoires et même menacer votre personne ou la santé des autres !



Le symbole « information » vous indique un paragraphe contenant des trucs et astuces ou des informations de fond. Il n'est pas essentiel de tout comprendre, mais cela reste tout de même très utile.

SOMMAIRE

1. Le module d’extension RP6 CCPRO M128

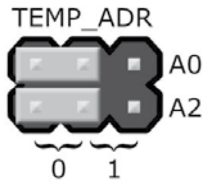
Grâce au module d’extension RP6 CCPRO M128 vous pouvez ajouter à votre système robot RP6 un module performant C-Control PRO MEGA128 de Conrad Electronic (code article 198219). Ce module comporte un contrôleur Atmel ATMEGA128 avec une Flash ROM 128 Ko, un SRAM de 4 Ko et de nombreux ports I/O. Il existe une mémoire d’extension sur le module d’extension, qui augmente la mémoire vive à 64 Ko et permet ainsi l’exécution d’importants algorithmes, comme par exemple, une simple préparation d’itinéraire. Le pilotage d’un hardware complexe est également possible.

Interruptions :



Ce bloc jumper peut être utilisé pour brancher PE6 et PE5 avec les câbles d’interruption du XBUS. Cela peut par exemple être utilisé pour éviter de constantes interrogations (« Polling ») du Slaves I²C. Au lieu de cela celles-ci mettent un des câbles d’interruption à un niveau Low lors de modifications d’état. Sur le schéma, PE6 et INT3 sont reliés, ainsi que PE5 et INT1, **il s’agit d’une configuration standard ! INT1 doit être relié avec PE5, afin que les programmes d’exemple fonctionnent correctement !** Mieux vaut ne pas brancher PE6 avec deux câbles d’interruption en même temps, même si cela est possible avec le bloc jumper !

Adresse du capteur de température bus I²C :



Deux bits (A0 et A2) de l’adresse du capteur de température peuvent être modifiés afin de pouvoir corriger d’éventuels conflits (c’est-à-dire d’adresses identiques) avec d’autres produits sur le bus I²C.

Tenez compte de la fiche technique du capteur de température !

L’adresse du capteur de température correspond en écriture binaire à : 1001[A2]A[A0]0.

A vrai dire, cette adresse n’est longue que de 7 bits, soit 100100 --> 0x48. Le bit écriture/lecture doit être cependant directement incorporer dans le programme (sera placé automatiquement), d’où 10010000--> 0x90 (144 décimales).

L’image indique le **réglage standard** de l’adresse sur 0x90.

D’autres adresse possibles : 0x92, 0x98 et 0x9A.

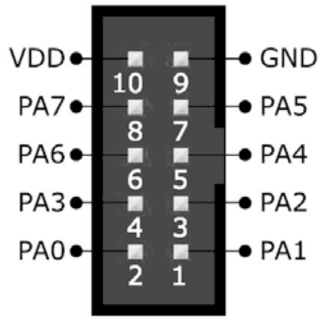
Si vous ne souhaitez pas installer le LCD standard, vous pouvez assembler votre propre câble à 14 pôles en se basant sur l'affectation des broches ci-contre.

Les lignes D0, D1, D2, D3, RW sont fermement branchées avec GND, puisque nous exploitons le LCD seulement en mode 4 Bit et qu'il n'y a pas besoin d'y lire quelque chose (RW est également sur la masse).

Veillez absolument à une affectation correcte des broches et à ne pas brancher les connecteurs à l'envers !

Les désignations des broches varient d'un fabricant à un autres, mais elles sont généralement identiques à celles que nous avons utilisées ici et vous pouvez brancher les broches 1 :1 avec l'écran !

PORTA et PORTC



Les connecteur pour PORTA et PORTC ne sont PAS équipés. Ils ne sont qu'optionnels, au cas où vous auriez réellement besoin de nombreuses broches I/O, mais que vous pouvez renoncer au SRAM externe. Vous pouvez le désactiver sur un jumper non équipé lui aussi (le branchement de ce jumper est placé sur l'arrière de la platine via piste conductrice – on peut facilement le sectionner à l'aide d'un cutter, et ensuite dessouder le jumper).

L'affectation des broches du PORTC est imprimée sur la carte mère. Vous pouvez, par exemple, dessouder une barrette à un seul rang de broches et à 9 pôles si besoin est.

Jumper

Vous avez certains jumper sur le module afin de configurer les composants sur le module.

En plus de ses composants centraux, le module comporte également un capteur de température de 12 Bit, un piezo pour la création d'un signal sonore, 5 LED et un port pour écran LC, sur lequel vous pouvez directement brancher un LCD texte standard.

Plusieurs possibilités d'extension sont disponibles grâce à 3 branchements pour des servos, 16 I/O libres (donc 8 canaux convertisseurs analogique/numérique (ADC)), un UART (une deuxième interface séquentielle), un socle DIL8 pour un bus EEPROM I^2C de la série 24LCxxx, ainsi qu'un branchement bus SPI pour la commande d'autres hardwares tels qu'un registre à glissement, ADC, DAC, Flash-ROM (cartes mémoire) ou autre. Les interfaces sont disponibles sur des connecteurs standard, à 10 pôles, et peuvent être facilement reliés à la plaque perforée de la platine d'expérimentation.

En désactivant quelques composants sur la carte-mère, vous pouvez libérer jusqu'à 19 autres ports I/O et 3 sorties de commutation (16 ports I/O reviennent au SRAM, les branchements pour une autre utilisation de ces port doivent absolument être équipés d'un connecteur enfichable).

Comme environnement de développement, vous pouvez utiliser le CCPRO IDE de Conrad Electronic. Il est très facile à utiliser et propose plusieurs fonctions confortables. Il y a deux langages de programmation différents au choix : Basic et Compact C. Le Basic est plus facile à apprendre pour les débutants que le C, mais a une étendue de fonctions identique. La bibliothèque de fonctions du CCPRO Unit est importante et prend même le multithreading en charge. Référez-vous à la documentation détaillée du C-Control Pro pour plus d'informations !

Avant que vous ne mettiez en route votre RP6 CCPRO M128, vous devez absolument vous familiariser avec le robot en testant tous les programmes d'exemple SANS que le module d'extension ne soit monté dessus. Il faut considérer ce mode d'emploi comme un ajout à celui du RP6 ET du C-Control PRO. Dans tous les cas, veuillez lire ce manuel avant d'utiliser votre RP6-M128 ! Vous pouvez survoler le chapitre sur la programmation si besoin est, et dans ces cas là lire le mode d'emploi du CCPRO Unit. Veillez cependant à ce que les programmes d'exemple du CCPRO Unit n'ont pas été écrits spécialement pour le RP6. Seule le code de programme de ce mode d'emploi et dans les programmes d'exemple est directement exécutable sur le module d'extension. Tous les autres programmes doivent être auparavant légèrement adaptés (avant tout aux autres affectations de broches).

Indication importante pour les débutants : Les programmes écrits pour le RP6 CCPRO M128 ne fonctionnent ABSOLUMENT PAS correctement sur le micro contrôleur de l'unité de base et vice et versa (processeurs, disposition des broches et fréquence élémentaire totalement différents) !

1.1. Support technique

(img 4)

Pour toute question ou problème, vous pouvez joindre notre Support Team comme suit sur Internet (avant de nous contacter, **veuillez lire attentivement et entièrement ce mode d'emploi !** La plupart des réponses à vos questions se trouvent dedans !) :

- sur notre forum : `http://www.arexx.com/forum`

- par e-mail : `info@arexx.nl`

- adresse postale :

  AREXX Engineering
  Nervistraat 16
  8013 RS ZWOLLE
  PAYS-BAS

- Pour toutes les informations de contact, les mises à jour logiciel et autres informations, rendez-vous sur notre page d'accueil :

`http://www.arexx.com/`

et sur la page d'accueil du robot :

`http://www.arexx.com/rp6`

et sur la page d'accueil du C-Control :

`http://www.c-control.de`

ainsi que sur le forum (non officiel) C-Control :

`http://ccpro.cc2net.de`

1.2. Contenu

Vous devez trouver les articles suivants dans la boîte du RP6 CONTROL M32 :

• Le module RP6 CCPRO M128

• CD-ROM avec le logiciel et la documentation

• 4 x boulon d'écartement 25 mm M3

• 4 x vis M3

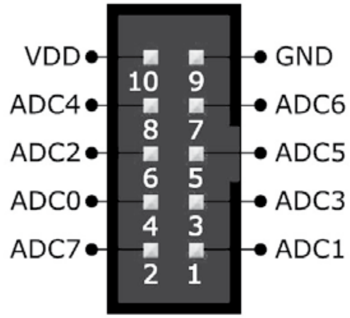
• 4 x écrou M3

• 2 x câble ruban 14 pôles

Le logiciel et le mode d'emploi PDF se trouvent sur le CD-ROM. Les versions actualisées du logiciel et ce mode d'emploi se trouvent également sur notre page d'accueil ! Le logiciel pour le C-Control PRO est disponible sur http://www.c-robotics.com ou http://www.c-control.de !

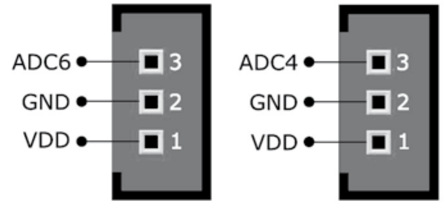
ATTENTION : il y a plusieurs variantes de servos ! Vérifiez auparavant les affectations des broches et faites attention à la polarité !

Les canaux ADC :

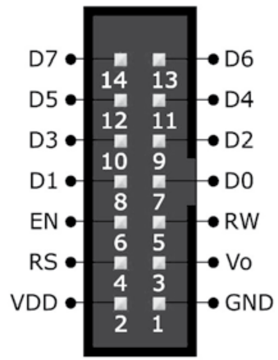


Les 8 canaux ADC libres (qui sont bien évidemment utilisables comme broches I/O) sont disponibles avec la tension d'alimentation sur le connecteur ADC à 10 pôles.

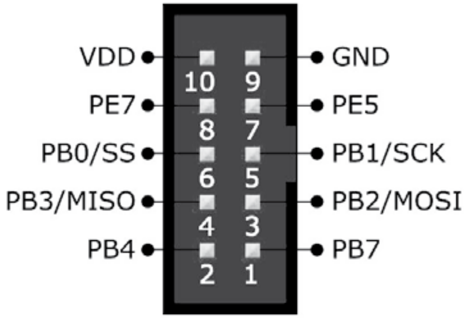
Deux des ADC (ADC4 et ADC6) sont en plus disponibles sur deux connecteurs à 3 pôles pour brancher, par exemple, directement des capteurs de distance IR analogiques.



Port LCD :



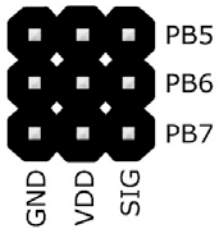
Bus SPI et ports I/O :



Vous pouvez brancher jusqu'à 5 puces compatibles SPI sur le bus SPI (et même plus avec un décodeur d'adresse externe). Le bus SPI peut être utilisé pour un registre à glissement rapide, EEPROM, mémoire FLASH, convertisseur N/A et A/N, etc. 5 broches I/O sont disponibles (PB0/SS est réservé pour le bus SPI lorsqu'il est en mode Slave SPI et peut sinon être utilisé comme un I/O), qui peuvent être utilisées pour les signaux « Chip select » ou tout simplement comme des broches I/O tout à fait normales. Il s'agit de PB0, PB4, PB7, PE5 et PE7.

PB1, PB2 et PB3 ne peuvent PAS être utilisées comme broches I/O normales, un registre à glissement est déjà connecté au module afin de commander le LCD et certaines LED (lorsque vous n'avez pas besoin de ces composants, une utilisation de ces broches comme I/O normales est envisageable. Attention au schéma de connexion !). PB7 est disponible en plus sur l'un des branchements servo, vous ne devez donc l'utiliser que pour un servo OU une ligne Chipselect.

Les servos :



Comme nous l'avons déjà mentionné, PB5, PB6 et PB7 sont disponibles aux branchements servos à 3 pôles. Vous pouvez brancher ici directement des servos avec une affectation des connecteurs adaptée.

1.3. Caractéristiques et données techniques

Cette partie vous offre un aperçu de ce que le RP6 CCPRO M128 vous offre et sert en même temps d'introduction de certains termes et appellations des composants du module.

Caractéristiques, composants et données techniques du RP6 CCPRO M128 :

- **Logement pour le module puissant C-CONTROL PRO MEGA128 avec un microcontrôleur 8 Bit Atmel ATMEGA128**
  - ◊ Cadence de 14,7456Mhz (dite quartz de taux de Baud pour atteindre le taux de bit le plus exact avec l'interface séquentielle.
  - ◊ Mémoire : 128 Ko Flash ROM, 4Ko SRAM interne et 64 Ko de SRAM externe pour l'extension sur le module, 4 Ko EEPROM.
  - ◊ Programmable en Basic et CompactC.
  - ◊ …retrouvez plus d'infos techniques sur la documentation du CCPRO Unit) !
- **Connecteurs d'extension Bus I²C**
  - ◊ Peut contrôler tous les Bus Slaves I²C.
  - ◊ Le MEGA128 peut être utilisé comme Master ou Slave. Généralement, il vaut mieux l'utiliser comme Master pour un contrôle complet du robot (le contrôleur de la carte mère se charge en effet de la régulation de la vitesse du moteur, ACS, IRCOMM, surveillance de l'accu, etc. de façon autonome et décharge ainsi le contrôleur du module d'extension).
- **Capteur de température bus I²C 12 Bit**
  - ◊ Afin de mesurer précisément la température environnante.
  - ◊ Résolution au choix de 0,5 à 0,0625°C. Exactitude la mesure d'env. 1°C.
- **Socle pour Bus EEPROM I²C du type 24(L)Cxxx (non fourni)**
  - ◊ Les données sont conservées après l'arrêt sur l'EEPROM.
  - ◊ Disponible jusqu'à une capacité de 1 Mbit (= 128 Kbyte) – idéal, entre autres, pour les enregistreur de données.
  - ◊ 1 million de fois réinscriptible
- **Signaleur piezo**
  - ◊ Pour créer des sons et des mélodies simples
  - ◊ Signaleur pour faire part d'une erreur ou d'un changement de statut, par exemple
  - ◊ Peut être désactivé – le port I/O est alors disponible pour le connecteur.
- **5 LED de statut**
  - ◊ pour la représentation des états d'un programme ou d'un capteur
  - ◊ Deux LED sont branchées à un port I/O, les trois autres à la sortie du registre à glissement. Vous pouvez désactiver les LED via le Jumper – les I/O ou les sorties de commutation sont alors disponibles pour d'autres applications.

• **Port Ecran LC**

◊ Pour le branchement d'un écran LC texte standard. De préférence des LCD compatibles HD44870, par exemple avec 16 x 2 ou 16 x 4 lignes. Nous vous conseillons toutefois de prendre les dimensions avant l'achat et de commander le matériel de montage adapté !

◊ L'écran peut afficher, par exemple, des messages textes, des menus, des statuts de programme ou encore des valeurs de capteur.

• **19 ports I/O disponibles pour le contrôle de votre circuit et de vos capteurs**

◊ **8** d'entre eux sont utilisables comme **canaux convertisseurs analogique/ numérique (ADC)**.

◊ **3** peuvent servir de **sorties PWM**, par exemple pour le contrôle de servos.

◊ Une **interface séquentielle supplémentaire (UART)** est disponible sur deux I/O.

◊ Vous pouvez libérer jusqu'à 19 ports I/O en désactivant les composants sur le module. 16 d'entre eux sont de toutes façons alloués pour le SRAM. Les connecteurs de ces ports I/O ne sont pas soudés au module, puisqu'ils ne sont utilisés qu'en cas d'exception. Le grand SRAM est généralement plus utile (Vous pouvez ajouter des I/O sur le port Expander du bus I²C si besoin est).

• **Jusqu'à 2 interrupteurs externes sur la connexion XBUS utilisables.**

• **Un bouton poussoir pour le démarrage du programme et d'autres fonctions.**

• **Branchement pour une interface USB RP6 pour le téléchargement de programme**

◊ Le téléchargement de programme fonctionne sur l'interface USB, comme pour le robot, mais dans tous les cas pas avec le logiciel RP6Loader mais directement sur le CCPRO IDE. Vous pouvez, si vous le souhaitez, utiliser la carte d'application CCPRO pour programmer le CCPRO Unit. Vous devez pour cela le changer de prise (cela n'est pas cependant pas nécessaire lorsqu'un LCD doit encore être monté).

Sont fournis également pour les langages Basic et CompactC 20 programmes d'exemple chacun, afin de vous permettre un apprentissage plus rapide.

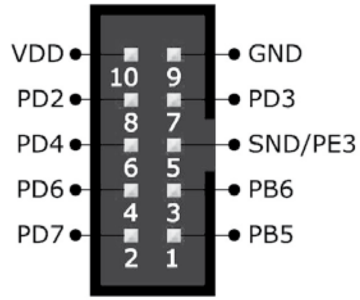
Vous pouvez trouver sur les pages web du robot et du C-Control PRO plus de programmes et de mises à jour, mis à disposition. Bien évidemment, vous pouvez aussi échanger votre propre programme sur Internet avec d'autres utilisateurs !

ANNEXE

A. Affectations des broches

Vous trouverez dans ce chapitre les affectations des broches des connecteurs et des blocs à souder les plus importants.

Ports I/O :



Certaines broches I/O libres et la tension d'alimentation de 5V sont disponibles sur les connecteurs I/O.

SND/PE3, PB5, PB6, PD2, PD3, PD4, PD6 et PD7.

PE3 (OC3A) est relié au signaleur piezo (SND), celui-ci peut être désactivé par Jumper afin de libérer cette broche I/O. PB6 et PB5 sont disponibles en plus pour des branchements sur la plage avant de la platine. Bien évidemment, ces broches ne peuvent être utilisées que pour une seule chose à la fois – soit comme broche normale I/O OU comme sortie servo. Ainsi, n'utilisez PAS en même temps les broches sur les deux connecteurs !

ATTENTION : Veuillez ne pas connecter la broche de tension d'alimentation avec un autre mode d'extension (par exemple un module d'expérimentation) à la broche de tension d'alimentation de ce connecteur, afin d'éviter toute boucle de masse.

4. Programmes d’exemple

Vous trouverez sur le CD respectivement 20 programmes d’exemple complets en CompactC et Basic. Les programmes d’exemple pour le module CCPRO sont détaillés en allemand.

Ces programmes d’exemple démontrent les fonctions fondamentales du module d’extension du RP6 CCPRO M128. Comme pour le robot, elles ne représentent en aucun cas une solution optimale, mais plutôt un point de départ pour vos propres programmes. Nous avons délibérément choisir de procéder ainsi, afin qu’il vous reste encore quelque chose à faire – il serait en effet ennuyeux, de tester seulement des programmes préfabriqués, qu’en dites vous ?

Il existe quelques programmes d’exemple pour le C-Control PRO sur Internet et dans le pack du CCPRO IDE. Vous devez cependant faire attention, à ce que les autres programmes d’exemples s’adaptent au matériel du RP6 CCPRO M128 – cela risque sinon de ne pas fonctionner (les problèmes les plus courants sont une affectation des broches différente, l’emploi de modules d’hardware déjà utilisés ailleurs, comme par exemple les minuteurs, etc.) !

Les programmes d’exemple se montent les uns sur les autres – c’est-à-dire que « Example_01

_HelloWorld » doit être regardé, testé et compris avant « Example_02_Bepper » ! De « Example_07_RP6_Sensors1 » jusqu’à « Example_09_Behaviour4 », un programme d’exemple complexe est monté petit à petit, dans lequel le robot peut esquiver des obstacles à l’aide de la technologie des capteurs infrarouges.

Attention, tous les exemples pour le RP6 ne sont pas forcément compréhensibles pour les débutants ! Il est recommandé de démarrer dans un premier temps avec un programme facile et d’avancer pas à pas. Les programmes d’exemple du C-Control PRO sont avant tout important pour une compréhension fondamentale et doivent être travaillés au préalable. Des essais avec l’hardware RP6 ne fonctionnent malheureusement que dans de rares cas, puisque les programmes pour la carte d’application du C-Control PRO ont été écrits (hardware et affectation des broches différents). Mais lire les commentaires et chercher à saisir ce qui est réellement fait est très utile pour la compréhension !

Les exemples complexes utilisent par ailleurs le multithreading mis à la disposition du CCPRO. Ici (et pour d’autres choses encore), il est recommandé de lire la documentation du CCPRO avant de comprendre le mode de fonctionnement de programmes d’exemple spécifiques du RP6 !

Vous pouvez utiliser l’exemple comme modèle pour votre propre programme – il existe d’ailleurs pour chaque programme d’exemple une version complète mais non commentée.

Nous avons atteint la fin de ce petit mode d’emploi additionnel. Vous pouvez désormais faire place à votre propre créativité, et ainsi écrire de nouveaux programmes ou placer de nouveaux capteurs sur le RP2, contrôlés par le RP6-CCPRO-M128 – ou bien mettre tout autre chose en place !

2. Montage du module d’extension



Vous devez tout d’abord enficher correctement le CCPRO MEGA128 Unit sur le module d’extension. Pour votre sécurité, veillez à ne manipuler le module qu’avec un objet mis à la terre, afin qu’il se décharge. Par exemple, un tuyau nu d’un corps chaud ou d’une pièce métallique d’un boîtier d’ordinateur (prise à 3 pôles !).

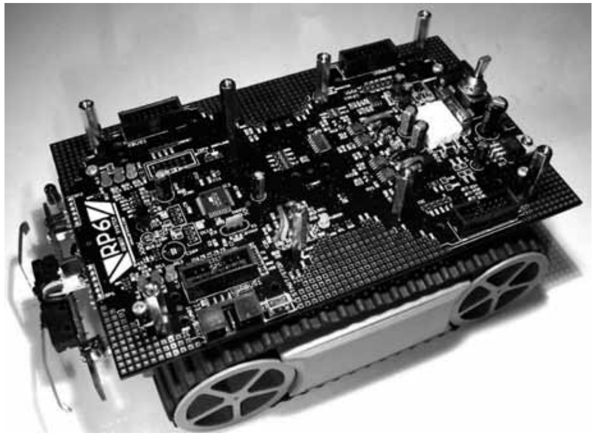
Veuillez contrôler AVANT le montage la position de tous les Jumper sur le module (cf. Annexe A) ! Ceux-ci doivent se trouver dans leur position standard.

Il n’y a qu’une orientation du module CCPRO adaptée (cf. image).Veillez bien à ce que toutes les broches soient correctement insérées dans les douilles de la platine !

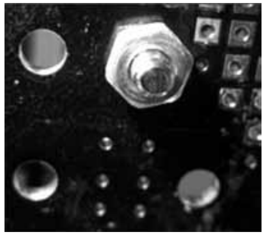
⚠ Le montage nécessite un peu de force, **mais veillez à ne pas appuyer trop fort !** Maintenez le dessous plat de la platine directement en dessous du module avec un ou deux doigts. **Attention :** les soudures tout autour sont légèrement pointues ! Veillez donc à ne toucher que la surface libre en dessous du module !

La façon dont vous devez fixer le module d’extension sur le robot dépend des autres modules d’extension que vous avez déjà, éventuellement, monté sur le robot.

Pour monter le module sur le robot, vous devez d’abord dévissez les 4 vis de la carte mère. Vous pouvez également, si vous le souhaitez, détacher prudemment un petit connecteur de la platine bumper, afin de soulever entièrement la carte mère. Cependant, cela n’est pas nécessaire, si vous vous servez de vos doigts pour atteindre ce qu’il y a en dessous de la carte mère et pour visser les boulons d’écartement avec les écrous M3.



Attention : Lorsque vous rebranchez le câble de la platine Bumper, maintenez l'arrière de la platine du capteur avec un doigt afin d'éviter qu'elle ne soit poussée trop fortement vers l'arrière ! Autrement, vous pouvez également dévissez les 2 vis de la platine Bumper et faire passer le câble…



Vous pouvez ensuite vissez les 4 boulons d'écartement 25mm M3 avec les écrous M3 dans les trous de fixation de la carte mère, comme représenté sur la photo.

Sur l'image au dessus, les 8 boulons d'écartement sont vissés, et également ceux du circuit imprimé du module d'extension !

Installez le module d'extension sur les boulons d'écartement et fixez le avec les 4 vis M3.

Enfin, connectez les deux câbles ruban – c'est fini.


```
printLCD(__STRING__)
printlnLCD(__STRING__, __LINE__, __POS__)
showScreenLCD(__STRING1__, __STRING2__)
printIntegerLCD(__INT__)
```

Ces macros fonctionnent de la même façon que ceux pour l'interface séquentielle. La position des caractères est bien évidemment très importante sur un LCD.

printLCD ne fait qu'initialiser un array temporaire avec un texte constant transmis, puis l'envoyer au LCD avec writeStringLCD.

printlnLCD appelle writeLineLCD.

ShowScreenLCD efface l'intégralité du contenu de l'écran et écrit ensuite les deux textes transmis à la ligne 1 et à la ligne 2. Cela n'est bien sur utile que pour les écrans à 2 lignes – si vous souhaitez utiliser un écran à 4 lignes ou plus, vous devez l'adapter en conséquence.

Il existe un équivalent à printInteger pour l'interface séquentielle avec printIntegerLCD. Avec ce macro, la valeur est envoyée comme une variable transmise en texte au LCD. Cela est très utile lorsque vous souhaitez visualiser les valeurs du capteur.

Vous pouvez trouver de nombreux exemples d'application du LCD dans les programmes d'exemple du CD-Rom ou de la page d'accueil du RP6. Le LCD est utilisé dans chaque programme d'exemple de façon plus ou moins intensive.

Ce chapitre ne sert que d'un grand aperçu des fonctions, qui sont disponibles comme extensions aux fonctions déjà intégrées du C-Control Interpreter dans la RP6CCLib. La façon dont vous devez les utiliser dans un programme complet est montrée de façon détaillée dans les programmes d'exemple.

Vous pouvez facilement modifier vous-même la RP6CCLib et ajouter de nouvelles fonctions si besoin. Il serait d'ailleurs préférable dans ce cas d'utiliser un nouveau fichier Basic ou C et de l'intégrer séparément. La façon dont cela fonctionne est bien évidemment démontrée dans les programmes d'exemple. Vous pourrez y créer une autre petite bibliothèque, qui continuera à vous simplifier dans l'exploitation du capteur et le contrôle du robot sur des bus I²C. Il y a également des fonctions d'aide telles que RP6_rotate ou RP6_move, avec lesquelles vous pouvez faire tourner le robot selon un certain angle, ou le faire avancer sur une distance précise. Les fonctions peuvent aussi réagir automatiquement lorsqu'il y a modification de données importantes de capteur (par exemple, lorsque l'ACS détecte un obstacle, une certaine fonction est appelée).

Nous n'allons pas parlé plus longtemps de ces choses complémentaires, car ces fonctions sont petit à petit ajouter dans les programmes d'exemple et décrites. Veuillez lire les commentaires dans les programmes d'exemple.

et

```
void writeCharLCD(char ch)
Sub writeCharLCD(ch As Char)
```

Les deux transmettent un byte au LCD, interpréter comme ordre de commande par writeLCDCommand et comme code caractère par writeCharLCD.

Avec la fonction suivante, le curseur texte sera placé à une position précise.

```
void setCursorPosLCD(byte text_line, byte pos)
Sub setCursorPosLCD(text_line As Byte, pos As Byte)
```

Exemple CompactC :

```
setCursorPosLCD(1, 5);       // Placer le curseur ligne 1, position 5
writeCharLCD (65);           // Ecrit un A à cette position
```

Exemple Basic :

```
setCursorPosLCD(1, 5)        ' Placer le curseur ligne 1, position 5
writeCharLCD (65)            ' Ecrit un A à cette position
```

```
void clearLCD()
Sub clearLCD()
```

Efface l'intégralité du contenu du LCD.

```
void setCursorPosLCD (byte text_line, byte pos)
Sub setCursorPosLCD(text_line As Bye, pos As Byte)
```

N'efface qu'une partie précise du LCD. Débutant à la ligne « text_line » à la position « pos », « lenght » caractères seront effacés.

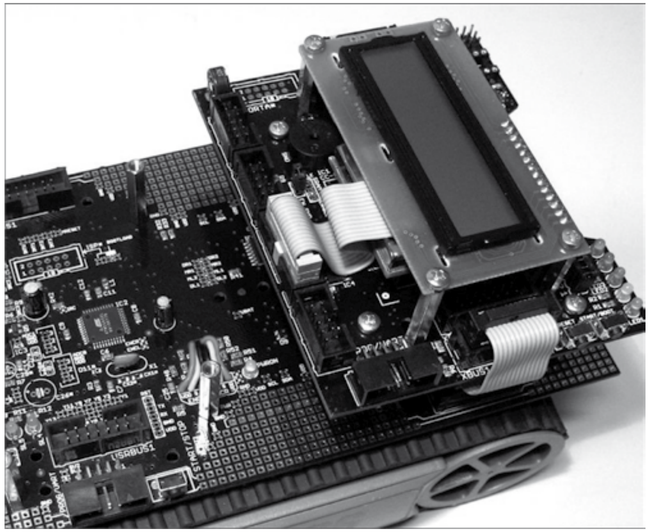
Si vous souhaitez envoyer plusieurs caractères l'un après l'autre au LCD, vous pouvez utiliser la fonction suivante.

```
void writeStringLCD(char text[])
Sub writeStringLCD(ByRef text As Char)
```

Si vous souhaitez insérer plusieurs caractères l'un après l'autre, et à une position précise, utilisez la fonction suivante.

```
void writeLineLCD(char text[], byte text_line, byte pos)
Sub writeLineLCD(ByRef text As Char, text_line As Byte, pos As Byte)
```

Pour toutes les fonctions String, il existe des macros qui simplifient l'application :

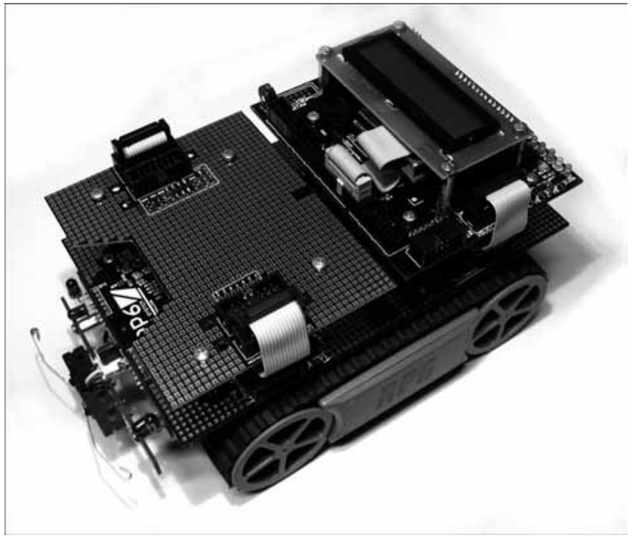


Nous vous recommandons de monter le RP6 CCPRO M128 à l'arrière du monceau d'extension du robot. Les deux branchements de programmation sont alors aussi accessibles depuis la même face du robot. Vous pouvez fixer à l'avant du robot les modules d'expérimentation fournis avec le robot (cf. photo ci-dessous pour avoir un exemple de configuration).

Si vous avez acheté un écran LC, vous devez d'abord le brancher et le monter au module d'extension avant de l'installer sur le robot. En fonction du format d'écran, vous allez devoir fabriquer un adaptateur afin de fixer l'écran. Cela est surtout valable pour les écrans de grande taille. Les petits LCD 2 x 16 lignes d'un format standard peuvent être directement fixés dessus grâce à 2 voire 4 boulons d'écartements. Afin d'installer l'écran encore au-dessus du module CCPRO, vous avez besoin de 2 boulons d'écartements 25 mm, de 2 vis et de 2 écrous au minimum. Pour les plus grands écrans, vous aurez besoin de matériel de montage nécessaire.

Le câble ruban à 14 pôles de l'écran LC standard 2 x 16 lignes est très flexible et peut être plié sans problème. Vous pouvez également installé l'écran d'une façon différente de celle que nous vous expliquons ici.

Vous pouvez également utiliser un autre écran LC texte compatible avec le contrôleur HD44780. Vous devez simplement souder un câble à l'écran. Veillez absolument à la bonne affectation des broches ! En cas d'inversion de polarité, l'écran et le contrôleur pourraient être gravement endommagés !



4 boulons d'écartement ne sont pas forcément nécessaires, comme sur la photo ! 2 suffisent amplement (tous les deux devant ou à l'arrière) pour fixer l'écran. Une fois le montage terminé, cela peut ressembler, par exemple, à la photo.

Vous pouvez aisément connecter les 3 connecteurs d'extension à 10 pôles avec les ports I/O libres et le bus SPI sur un petit câble ruban à 10 pôles avec une plaque perforée du module d'extension et utiliser alors les I/O et les ADC pour, entre autres, l'exploitation des capteurs. Cela fonctionne également avec les modules montés sur l'autre côté – les câbles ruban passent normalement facilement entre l'espace au milieu des deux monceaux de modules.

2.1. Test des fonctions

Maintenant vous pouvez effectuer un petit test des fonctions, afin de vérifier que le module d'extension fonctionne correctement.

Pour cela vous devez avant tout vous familiariser avec le CCPRO IDE (et bien évidemment l'installer comme cela est décrit dans le mode d'emploi du CCPRO. Vous pouvez utiliser le pilote USB du CD-ROM du RP6, référez-vous au mode d'emploi du RP6 pour en savoir plus sur l'installation) ! Afin d'effectuer le test, vous devez relier interface USB connectée à votre ordinateur via le câble ruban à 10 pôles au branchement PROG/UART du RP6 CCRPO M128 et démarrer le CCPRO IDE. Ouvrez ensuite le projet :

Cela doit ressembler par exemple à ce qui suit :

```
Thread_Lock(1);               // Important : Bloquer la commutation thread !
Port_WriteBit(PORT_FLASH, 1);// Activer la ligne CS
SPI_Write(10);                // Envoyer les données
Result = SPI_Read();          // Lire les données
SPI_WriteBuf(buffer, 32);     // Envoyer 32 Bytes
SPI_WriteBuf(buffer, 16);     // Envoyer 16 Bytes
Port_WriteBit(PORT_STR, 0);   // Désactiver à nouveau la ligne CS
Thread_Lock(0);               // Attention : débloquer la commutation thread !
```

Cela n'est bien évidemment pas un exemple complet, il doit être adapté à chaque slave !

Vous devez faire là bien attention aux fiches de données du constructeur. Il est important de bloquer la commutation thread l'intégralité du temps d'accès. Sinon deux threads tenteraient en même temps d'accéder au bus SPI, ce qui ne fonctionne pas. Vous devez particulièrement faire attention ici, car chaque accès au LCD et aux LED exige un accès SPI !

3.8. Ecran LC

L'écran LC est idéal pour l'affichage des valeurs du capteur et des messages d'état. Il peut être très utile pour obtenir des informations sur l'état du programme actuel, surtout quand le robot n'est pas relié à un ordinateur.

Le LCD n'est pas branché comme sur la carte d'application CCPRO, les fonctions LCD internes ne pouvant être utilisées directement. Cela devrait toutefois être possible dans une prochaine version de l'interpréteur CCPRO.

En tout premier, vous devez bien évidemment initialiser le LCD :

```
void RP6_initLCD(void)
Sub void RP6_initLCD()
```

Le RP6_CCPRO_Init() le fait normalement déjà – vous n'avez donc pas besoin, normalement, de le faire à nouveau.

Le LCD fonctionne en mode 4 bit. Ainsi, vous devez appeler 2 fois la fonction suivante pour transmettre un byte de données complet au LCD.

```
void setLCDD(byte lcdd)
Sub setLCDD(lcdd As Byte)
```

Ces deux fonctions suivantes le font également :

```
void writeLCDCommand(byte cmd)
Sub writeLCDCommand(cmd As Byte)
```

contenus dans la RP6CCLib ! Comme celles-ci ne sont pas disponibles très longtemps dans l'interpréteur CCPRO, elles sont rapidement écrites. Elles doivent être enregistrées peu après dans la documentation CCPRO.

```
void SPI_Enable(byte ctrl)
```

Initialisez le module hardware SPI avec le byte de configuration « ctrl ». La valeur standard pour un module d'extension CCPRO est de 0x50, vous devez alors fixer les bits SPE (Enable) et MSTR (Master Modus) dans le registre de configuration (cf. fiche de données MEGA128).

La cadence d'horloge est réglée sur la moitié de celle de la CPU, soit 7.3728 MHz.

```
byte SPI_Read(void)
```

Lit un byte d'un slave SPI tout juste activé.

```
void SPI_Write(byte data)
```

Envoie un byte au slave SPI tout juste activé.

```
void SPI_ReadBuf(byte buf[], byte length)
```

Lit les bytes « length » d'un slave SPI tout juste activé et les enregistre dans un circuit « buf ».

```
void SPI_WriteBuf(byte buf[], byte length)
```

Envoie les bytes « length » du circuit « buf » au slave SPI tout juste activé.

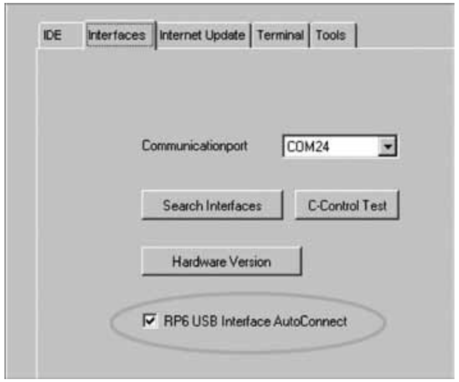
Pour un registre à glissement tel que le 74HC4094 utilisé sur un module d'extension CCPRO, vous devez activer une ligne « Strobe » après que vous ayez écrit les données.

Par exemple :

```
Thread_Lock(1);              // Important : Bloquer la commutation thread !
SPI_Write(external_port);    // Envoyer les données
Port_WriteBit(PORT_STR, 1);  // Activer la ligne strobe
delayCycles(5);              // très courte pause
Port_WriteBit(PORT_STR, 0);  // Désactiver à nouveau la ligne strobe
Thread_Lock(0);              // Attention : débloquer la commutation thread !
```

Cela s'effectue de la même façon que le RP6CCLib. Pour les autres slaves SPI, comme par exemple les puces mémoire Flash ou les EEPROM, il est plus généralement requis de mettre la ligne Chipselect avant l'accès BUS et de la désactiver ensuite.

<RP6_CCPRO_Examples>/CompactC/RP6_CCPRO_SELFTEST/RP6_CCPRO_Selftest.cprj



du CD et laissez le compilateur le traduire (la flèche bleu en haut dans la barre d'outils).

Vous devez choisir la bonne interface dans les réglages (COMxx, ne PAS choisir USB0, celui-ci ne fonction qu'avec la carte d'application CCPRO) et éventuellement cocher la case pour l'activation de « RP6 USB Interface AutoConnect » ! C'est important afin que le CCPRO IDE passe automatiquement en mode Bootload. Ainsi vous n'avez plus besoin d'appuyer sur deux boutons du module à chaque téléchargement de programme.

Si le CCPRO Unit n'est pas reconnu au premier essai, cliquez à nouveau sur le bouton « recherche d'interfaces » et assurez-vous que le CCPRO Unit est correctement passé en mode bootload (« Conrad C-Control Pro 2008 » doit apparaître dans la fenêtre d'édition). Dans le cas cela ne se passe pas automatiquement, vous pouvez appuyer sur les deux boutons du module CCRPO, dans l'ordre qui suit afin de passer le module en mode Bootloard : Appuyez sur START/BOOT et restez appuyé, appuyez sur RESET, lâchez RESTER et seulement après relâchez le bouton START/BOOT.

Lorsque le module CCPRO est en mode Bootload, la LED jaune s'allume de façon permanente. Vous pouvez alors charger le programme dans le CCPRO Unit. Après la réussite du téléchargement, démarrez le programme soit par pression sur le bouton Start du module CCPRO ou du robot lui-même. Vous devez d'abord appuyer une fois sur Reset, si jamais le module se trouve encore dans le mode Bootload (ou cliquez sur le symbole « éclair » dans la barre d'outils du CCPRO IDE). La LED verte est sensée clignoter lorsque le module est prêt à démarrer le programme chargé. Cela ne fonctionne cependant que lorsque vous utilisez le RP6CCLib ! Il y a en effet un sous-programme qui permet de démarrer tous les contrôleurs du RP6 en même temps.

Le signaleur Piezo doit émettre un court son directement après avoir appuyé sur le bouton Start. La LED s'éclaire et un texte est affiché sur le LCD « RP6 CCPRO M128 ».

Des messages d'état sont transmis à l'interface séquentielle, qui peuvent se voir sur le CCPRO IDE (ou au terminal du RP6Loader). **Faites bien attention ici aux messages d'erreur possibles !**

Lorsque l'écran n'affiche rien ou seulement deux lignes de carrés noirs, mais que les LED clignotent et que la connexion avec le CCPRO IDE fonctionne, vous devez régler le contraste de l'écran (ou bien l'affectation des broches de l'écran est mauvaise…). Cela se fait avec le potentiomètre R10 à l'arrière de la platine.

Ajustez le contraste avec le potentiomètre R16 sur la platine. Il peut être réglé grâce à un petit tournevis plat. Vous pouvez également utiliser un petit tournevis cruciforme, mais vous rencontrerez des difficultés à le positionner par rapport au potentiomètre.

Tournez le potentiomètre alors que le robot est allumé, jusqu'à ce quelque chose de distinct apparaisse sur l'écran. Redémarrez alors le programme d'autotest !

Lorsque le programme d'autotest est terminé, le message « SELFTEST OK ! » doit apparaître sur l'écran LC.

L'édition du programme doit à peu près ressembler à ce qui suit :

```
     ________________________
      \| RP6  ROBOT SYSTEM |/
       \_-_-_-_-_-_-_-_-_-_/
  RP6 C-Control PRO M128 Selftest
  Writing Text to LCD...
  Testing Beeper...
  Testing LEDs...
  Testing Temperature Sensor...
Temperature: 19.6250°C --> OK
Temperature: 19.6875°C --> OK
Temperature: 19.8750°C --> OK
Temperature: 19.8125°C --> OK
Temperature: 19.8750°C --> OK
[...]
Temperature: 19.8125°C --> OK
Temperature: 19.6250°C --> OK
 Testing External 64K Memory...
MEM PATTERN: 17 - Writing...
Reading...
OK
MEM PATTERN: 34 - Writing...
Reading...
OK
MEM PATTERN: 68 - Writing...
[...]
MEM PATTERN: 64 - Writing...
Reading...
OK
```



```
// Lecture valeur mesurée:
newline();
print ("Temperarute : ");
Str_WriteFloat (temperature,4,result,0);
Serial_WriteText(0,result);
print("°C");
newline();
```

Exemple Basic :

```
Dim temperature As Single ' point de flottement variable pour la valeur de
la température
Dim result(32) As Char
Dim temp_low As Integer

' [...]

' Convertir valeur mesurée en valeur « Float » (mesure de 12 bits):
temp_low = getTemperatureLow ()
If Not temp_low And 128 Then
   temp_low = (temp_low And 63) – 127
Else
   temp_low = temp_low And 63
temperature = temp_low + (0.0625 * (getTemperatureHigh ()>>4))

' Lecture valeur mesurée:
newline(
print ("Temperature: ")
Str_WriteFloat (temperature,4,result,0)
Serial_WriteText(0,result)
print("°C")
newline()
```

La valeur mesurée sur l'interface sérielle est donnée dans les deux programmes d'exemple.

3.7. Bus SPI

Sur le bus SPI (= Serial Peripheral Interface), vous pouvez connecter différents composants. Le convertisseur analogique/numérique, le convertisseur numérique/analogique, les EEPROM, la mémoire Flash, des cartes mémoire, des écrans LC ou un registre à glissement entre autres.

L'écran LC optionnel du module d'extension CCPRO est par exemple relié à un registre à glissement 8 Bit, avec 3 des LED de statut. Nous vous parlerons des fonctions pour la commande de l'écran dans le prochain paragraphe.

L'accès au bus SPI est très simple, vous devez simplement commuter la ligne de chaque slave sur le niveau « high » et vous pouvez alors lire les données et les écrire.

Attention : Seule une ligne Chipselect doit être activée à la fois ! Sinon, la lecture des données fonctionne mal, puisque des données sont envoyées à deux slaves.

Les fonctions SPI sont des fonctions internes « interpréteur », et ne sont donc pas

Pour lire ensuite le capteur, nous allons utiliser cette fonction :

```
void TCN75_read(byte adr)
Sub TCN75_read(adr As Byte)
```

Exemple CompactC :

```
TCN75_run(TCN75_ADR, TCN75A_CONFIG_RES_12) ; // Démarrage des mesures
AbsDelay(250) ;                              // Pause
TCN75_shutdown(TCN75_ADR) ;                  // Arrêt des mesures
TCN75_read(TCN75_ADR) ;                      // lecture...
```

Exemple Basic :

```
TCN75_run(TCN75_ADR, TCN75A_CONFIG_RES_12)   ' Démarrage des mesures
AbsDelay(250)                                ' Pause
TCN75_shutdown(TCN75_ADR)                    ' Arrêt des mesures
TCN75_read(TCN75_ADR)                        ' lecture...
```

Cela vous permet de lire la valeur de la température actuelle. Elle doit bien évidemment être retravaillée par la suite...

Après que la valeur de la température actuelle ait été lue, vous pouvez accéder aux « Low » et « High » Byte de la valeur mesurée avec les fonctions :

```
byte getTemperatureHigh(void)
Sub getTemperatureHigh() As Byte
```

et

```
byte getTemperatureLow(void)
Sub getTemperatureLow() As Byte
```

Ces deux bytes ne contiennent pas la valeur du capteur dans le format adapté – vous pouvez la convertir en un chiffre de point d'écoulement.

Exemple CompactC :

```
Float temperature;
Char result[32];
Int temp_low;

// Valeur mesurée à convertir en valeur « Float » (mesure de 12 bits):
temp_low = getTemperatureLow ();
if( !temp_low & 128)
   temp_low = (temp_low & 63) – 127;
else
   temp_low = temp_low & 63;
temperature = temp_low + (0.0625 * (getTemperatureHigh ()>>4));
```



```
MEM PATTERN: 128 - Writing...
 Testing I2C Communication...
##########################################################
Selftest finished successfully!
```

Les valeurs mesurées de température seront bien évidemment différentes et l'édition légèrement plus courte.

3. Programmation

Comme il existe déjà une documentation détaillée pour toutes les applications et la programmation pour le CCPRO Unit, nous allons vous expliquer ici directement les fonctions spécifiques du RP6. Nous n'allons pas aborder ici sur les bases des langages CompactC ou BASIC ni comment contrôler les ports un à un, les ADC, l'interface séquentielle ou travailler avec les threads ! Tout cela est détaillé dans le mode d'emploi du C-Control PRO et décrit dans la fonction d'aide existante du CCPRO IDE.

Nous allons vous décrire les fonctions de la bibliothèque RP6 CCPRO (ou « RP6CCLib »). Pour chaque fonction, nous allons vous donner quelques codes d'exemple en CompactC et BASIC. Ces prototypes de fonction sont listés dans la notation C et BASIC. Vous trouverez plus d'exemples détaillés sur le CD-ROM.

3.1. Consulter le bouton start et démarrage du programme

```
void RP6_waitForStart(void)
Sub RP6_waitForStart()
```

Cette fonction est importante pour synchroniser le démarrage du programme avec d'autres contrôleurs du RP6. Les contrôleurs de votre programme doivent, si possible, démarrer en même temps et vous devez pouvoir contrôlé l'exécution du programme, en le stoppant sans éteindre complètement le robot.

Cette fonction attend d'abord un signal de départ précis. Cela peut être une pression sur le bouton Start sur le module RP6 CCPRO M128 ou un signal reçu d'un autre contrôleur (par exemple de la carte mère) sur le bus I²C. Lorsque vous appuyez sur le bouton Start, cela est signalé aux autres contrôleurs via le bus I²C.

3.2. Initialisation

```
void RP6_CCPRO_Init(void)
Sub RP6_CCPRO_Init()
```

Vous devez toujours appeler cette fonction à la place du RP6_watForStart. Elle initialise le module hardware et appelle ensuite le RP6_waitForStart.

Une structure de base de programme typique pour le RP6 CCPRO M128 peut ressembler à ce qui suit :

Exemple en CompactC :

```
1  #include "../../RP6CCLib/RP6CCLib.cc"
2
3  void main(void)
4  {
5      RP6_CCPRO_Init(); // Initialisierung – IMMER ALS ERSTES AUFRUFEN!
6
7      // [...] Programmcode...
8
9      while(true)     // Endlosschleife
10     {
11         // [...] Programmcode...
12     }
13 }
```

Exemple en Basic :

```
1  #include "../../RP6CCLib/RP6CCLib.cbas"
2
3  Sub main()
4         RP6_CCPRO_Init() ' Initialisierung – IMMER ALS ERSTES AUFRUFEN!
5
6         ' [...] Programmcode...
7
8         Do While True     ' Endlosschleife
9                ' [...] Programmcode...
10        End While
11 End Sub
```

Ces deux programmes sont par leur fonctionnalité identique. Chacun peut décider de lui-même s'il préfère programmer en BASIC ou en CompactC.

Indication : Vous pouvez intégrer le RP6_CCPRO_lib (ligne 1 dans l'exemple) à un paramètre de projet. Il est toutefois plus intéressant d'utiliser une indication de chemin relative plutôt qu'un chemin absolu (« ../../lib », deux niveaux répertoire supérieurs). Ainsi vous pouvez déplacer sans problème le projet avec la bibliothèque ou l'utiliser

I2C_REG_ADC_MOTOR_CURL_L I2C_REG_ADC_MOTOR_CURL_H	17, 18	Valeur ADC mesurée capteur de courant gauche (High et Low Byte)
I2C_REG_ADC_MOTOR_CURR_L I2C_REG_ADC_MOTOR_CURR_H	19, 20	Valeur ADC mesurée capteur de courant droite (High et Low Byte)
I2C_REG_ADC_UBAT_L I2C_REG_ADC_UBAT_H	21, 22	Valeur ADC mesurée tension batterie (High et Low Byte)
I2C_REG_ADC_ADC0_L I2C_REG_ADC_ADC0_H	23, 24	Valeur ADC mesurée, Canal ADC libre 0 (High et Low Byte)
I2C_REG_ADC_ADC1_L I2C_REG_ADC_ADC1_H	25, 26	Valeur ADC mesurée, Canal ADC libre 1 (High et Low Byte)
I2C_REG_RC5_ADR	27	Adresse et toggle bit des dernières données RC5 réceptionnées par paquets
I2C_REG_RC5_DATA	28	Données des dernières données RC5 réceptionnées par paquets
I2C_REG_LEDS	29	Etat du statut LED

3.6.3. Capteur de température

Il existe déjà des fonctions prêtes pour le capteur de température connecté à la platine par bus I²C. Avec :

```
TCN75_write_cfg(byte adr, byte config)
Sub TCN75_write_cfg(adr As Byte, config As Byte)
```

vous pouvez généralement écrire dans le registre de configuration du capteur (vous trouverez plus d'informations dans la feuille de données du capteur TCN75. Certains réglages ont déjà été mis dans la bibliothèque en tant que constantes).

Le code est plus visible lorsque les macros suivants sont utilisés :

```
TCN75_run(ADR, CONFIG)
```

et

```
TCN75_shutdown(ADR)
```

pour démarrer ou stopper l'utilisation. Ces macros appellent cependant TCN75_write_cfg. AvecTCN75_run vous pouvez transmettre en plus de l'adresse des capteurs l'octet de configuration.

Exemple Basic :

```
// Lire la valeur du capteur de tension :
Dim messageBuf (3) As Byte; // Buffer pour la réception de données
RP6-readRegisters(RP6_BASE_ADR, I2C_REG_ADC_UBAT_L, messageBuf, 2)
// La valeur mesurée est composée de 2 octets, vous devez les assembler en
// une valeur mesurée :
adcBat = (messageBuf (1) << 8) | (messageBuf (0)) ;
// (l'octet de valeur supérieure de 8 points est glissé à gauche et à la
// place est écrit l'octet de valeur inférieure sur une ODER combinaison)
// Maintenant vous pouvez donner la valeur :
print (« Battery Sensor Value : »)
printInteger (adcBat)
println (« »)
// Il s'agit ici non pas d'une valeur de tension directe, mais d'une
valeur
// mesurée grossière du convertisseur analogique numérique. Vous pouvez
// encore la convertir afin d'obtenir la véritable valeur (par 102.4
// parties).
```

Le programme standard RP6_I2C_Slave à 30 registres à sa disposition, avec lesquels vous pouvez lire les actuelles valeurs des capteurs et les états des programmes.

Nom du registre	#	Description
I2C_REG_STATUS1	0	Statut registre 1 Bit 0: batLow; 1: bumperLeft; 2: BumperRight; 3: RC5reception; 4: RC5transmitReady; 5: obstacleLeft; 6: obstacleRight; 7: driveSystemChange
I2C_REG_STATUS2	1	Statut registre 2 Bit 0: powerOn; 1: ACSactive; 2: watchDogTimer; 3: wdtRequest; 4: wdtRequestEnable;
I2C_REG_MOTION_STATUS	2	Statut registre pour actionnement système Bit 0: movementComplete; 1: motorsOn; 2: motorOvercurrent; 3+4: direction;
I2C_REG_POWER_LEFT	3	Valeur PMW gauche réglée actuellement
I2C_REG_POWER_RIGHT	4	Valeur PMW droite réglée actuellement
I2C_REG_SPEED_LEFT	5	Valeur mesurée encodeur gauche (/ 200 ms)
I2C_REG_SPEED_RIGHT	6	Valeur mesurée encodeur droite (/200 ms)
I2C_REG_DES_SPEED_LEFT	7	Valeur théorique de vitesse gauche
I2C_REG_DES_SPEED_RIGHT	8	Valeur théorique de vitesse droite
I2C_REG_DIST_LEFT_L I2C_REG_DIST_LEFT_H	9, 10	Distance accomplie gauche (High et Low Byte)
I2C_REG_DIST_RIGHT_L I2C_REG_DIST_RIGHT_H	11, 12	Distance accomplie droite (High et Low Byte)
I2C_REG_ADC_LSL_L I2C_REG_ADC_LSL_H	13, 14	Valeur ADC mesurée, capteur de luminosité gauche (High et Low Byte)
I2C_REG_ADC_LSR_L I2C_REG_ADC_LSR_H	15, 16	Valeur ADC mesurée, capteur de luminosité droite (High et Low Byte)

directement sur un autre ordinateur. Veillez bien à ce qu'il y est deux bibliothèques différentes : une pour BASIC (RP6CCLib.**cbas**) et une pour CompactC (RP6CCLib.**cc**).

3.3. Délivrer du texte

```
print(STRING)
println(STRING)
newline()
printInteger(Int)
```

Il n'y a pas de constante de texte dans CompactC et BASIC – C'est pour cela qu'un array toujours attribuer du texte en premier et le transmettre ensuite à la fonction Serial_WriteText.

Afin de conserver le programme clair et d'épargner un travail d'écriture, les macros RP6CCLib sont disponibles, qui prennent justement du texte en charge, jusqu'à 64 lignes (et valeurs numériques intégrées). « print » délivre le texte transmis et « println » ajoute en complément les caractères spéciaux «\r\n », afin que le terminal débute une nouvelle ligne.

Exemple Compact C :

```
Println (« bonjour le monde ! ») ;
Print (« Test1: ») ;
printInteger(x) ; // Donne la valeur de la variable x en tant que texte
ASCII
print (« ... Test2 ») ;
newline() ; // Nouvelle ligne
```

Exemple Basic :

```
Println (« bonjour le monde ! ») ;
Print (« Test1: ») ;
printInteger(x)  ' Donne la valeur de la variable x en tant que texte
ASCII
print (« ... Test2 ») ;
newline()    ' Nouvelle ligne
```

3.4. LED de statut

```
void setLEDs(byte leds)
Sub setLEDs(leds As Byte)
```

Les 5 LED de la carte mère peuvent être contrôlées à partir de cette fonction

Exemple CompactC :

```
setLEDs (LED1 | LED2 | LED5) ; // LED1, LED2 et LED5 allumées, LED3 et
LED4 éteintes.
setLEDs (0) ; // toutes les LED sont éteintes
setLEDs (31) ; // toutes les LED sont allumées (31 = binaire 11111)
```

Exemple Basic :

```
setLEDs (LED1 Or LED2 Or LED5) ' LED1, LED2 et LED5 allumées, LED3 et LED4
éteintes.
setLEDs (4) ' LED3 allumée
setLEDs (5) ' LED1 et 3 allumées (5 = binaire 00101)
```

Les LED1 et 2 sont directement branchées à des ports I/O normaux et peuvent donc être commandées comme des ports I/O normaux. Mais les LED3, 4 et 5 sont reliées, avec le LCD, à un registre à glissement au bus SPI. Cela permettant d'économiser des ports I/O, qui seront utilisés pour d'autres choses. Sinon le LCD bloquerait à lui seul 6 I/O, plus les 5 LEDS, cela fait 11 I/O au total. Grâce au registre à glissement, il n'y en a que 6 et le bus SPI peut rattacher du matériel extérieur supplémentaire.

3.5. Beeper

Le signaleur du RP6-M128 peut être contrôlé avec cette fonction :

```
void beep(word pitch, word time, word pause)
Sub beep(pitch As Word, time As Word, pause As Word)
```

Cependant cette fonction est bloquante – c'est-à-dire qu'elle fixe les hauteurs de son, bloque les laps de temps imposés, éteint le signaleur et marque une pause avant le deuxième laps de temps fixé. La fonction AbsDelay de la bibliothèque de fonction du CCPRO est utilisée ici. Les temps sont ici retranscrits en ms.

Exemple CompactC :

```
#define Tone_A1  262      // 440Hz
beep (Tone_A1, 100, 200)  // 100ms de son, 200ms de pause
beep (Tone_A1, 355, 422)  // 355ms de son, 422ms de pause
```

Exemple Basic :

```
#define Tone_A1  262      ' 440Hz
beep (Tone_A1, 100, 200)  ' 100ms de son, 200ms de pause
beep (Tone_A1, 355, 422)  ' 355ms de son, 422ms de pause
```

Cette fonction beep est principalement pensée pour l'édition de mélodies. Si vous souhaitez laisser d'autres choses fonctionner en parallèle, vous pouvez utiliser les threads. Veillez cependant à ne pas utiliser cette fonction en même temps dans deux threads différents (vous ne pouvez pas avoir un accès simultané à deux threads différents sur un même composant hardware) !

Cela ne fonctionne de toutes façons pas avec la fonction beep normal, car AbsDelay est utilisé ici. AbsDelay interrompt les éditions de tous les threads. C'est pourquoi il existe cette fonction spéciale :

```
void beep_t(word pitch, word time, word pause)
Sub beep_t(pitch As Word, time As Word, pause As Word)
```

qui utilise Thread_Delay au lieu de AbsDelay. L'inconvénient est que Thread_Delay n'a

Un programme d'exemple vous montre comment utiliser un ordre seul.

Pour écrire une valeur de 16 bit, comme par exemple pour l'ordre CMD_Move, vous devez partager cette valeur en 2 valeurs de 8 bit.

Exemple CompactC

```
Word distance; // Valeur de 16 bit qui doit être écrite
Distance = 4000;  // 4000 * 0.25mm = 1 mètre
Byte params[5] ;
params[0] = 80    // 10 cm/s
params [1] = FWD  // marche avant
params [2] = distance >> 8;   // distance au-dessus 8 bit
params [3] = distance & 0x0F; // en dessous de 8 bit
RP6_writeCommand_params(10, CMD_Move, params, 4);
```

3.6.2. Lire des données

Bien évidemment, vous ne transmettez pas seulement des ordres, vous pouvez également lire les données, comme par exemple les valeurs des capteurs. Vous pouvez lire une registre seul avec la fonction suivante :

```
byte RP6_readRegister(byte adr, byte reg)
Sub RP6_readRegister(adr As Byte, reg As Byte) As Byte
```

Vous pouvez lire plusieurs registres avec :

```
void RP6_readRegisters(byte adr, byte reg, char readBuffer[], byte reg_count)
Sub RP6_readRegisters(adr As Byte, reg As Byte,
                      ByRef readBuffer As Byte, reg_count As Byte)
```

et ils seront écrits sur l'array « readBuffer ».

Exemple CompactC :

```
// Lire la valeur du capteur de tension :
byte messageBuf [3] ; // Buffer pour la réception de données
RP6-readRegisters(RP6_BASE_ADR, I2C_REG_ADC_UBAT_L, messageBuf, 2) ;
// La valeur mesurée est composée de 2 octets, vous devez les assembler en
// une valeur mesurée :
adcBat = (messageBuf [1] << 8) | (messageBuf [0]) ;
// (l'octet de valeur supérieure de 8 points est glissé à gauche et à la
// place est écrit l'octet de valeur inférieure sur une ODER combinaison)
// Maintenant vous pouvez donner la valeur :
print (« Battery Sensor Value : ») ;
printInteger (adcBat) ;
println (« ») ;
// Il s'agit ici non pas d'une valeur de tension directe, mais d'une
valeur
// mesurée grossière du convertisseur analogique numérique. Vous pouvez
// encore la convertir afin d'obtenir la véritable valeur (par 102.4
// parties).
```

Le programme RP6I2CSlave connaît actuellement 12 ordres, permettant de contrôler entièrement le robot.

Ordre	Code	Description
`CMD_POWER_OFF`	0	Eteindre l'encodeur, le détecteur de courant et Power LED
`CMD_POWER_ON`	1	Allumer l'encodeur, le détecteur de courant et Power LED
`CMD_SETLEDS`	3	Régler les LED Paramètre 1 : Etat LED – 6 premiers bits Bit 0 = LED1, Bit 6 = LED6
`CMD_STOP`	4	Arrêter les moteurs et la régulation de la vitesse
`CMD_MOVE_AT_SPEED`	5	Bouger à une vitesse précise Paramètre 1 : vitesse gauche Paramètre 2 : vitesse droite Valeur de la vitesse à l'encodeur par niveau de 200ms
`CMD_CHANGE_DIR`	6	Modifier le sens de rotation du moteur Paramètre 1 : Direction – FWD, BWD, LEFT ou RIGHT (les constantes sont définies dans le RP6CCLib)
`CMD_MOVE`	7	Effectuer une distance précise Paramètre 1 : vitesse Paramètre 2 : direction, FWD ou BWD Paramètre 3 : High Byte de l'itinéraire accompli Paramètre 4 : Low Byte
`CMD_ROTATE`	8	Pour effectuer un anglé précis Paramètre 1 : vitesse Paramètre 2 : Direction LEFT ou RIGHT Paramètre 3 : High Byte de l'angle Paramètre 4 : Low byte de l'angle
`CMD_SET_ACS_POWER`	9	Régler l'énergie d'émission des ACS Paramètre 1 : énergie d'émission ACS_PWR_OFF, ACS_PWR_LOW, ACS_PWR_MED, ACS_PWR_HIGH
`CMD_SEND_RC5`	10	Emettre un signal RC5 avec l'IRCOMM Paramètre 1 : adresse et toggle bit Paramètre 2 : données
`CMD_SET_WDT`	11	Activer le minuteur Watchdog Paramètre 1 : true (vrai) ou false (faux)
`CMD_SET_WDT_RQ`	12	Activer la requête minuteur Watchdog Paramètre 1 : true ou false

un temps de pause que de 10ms exactement. C'est-à-dire que le programme ci-dessus aurait ressemblé à ça :

Exemple CompactC :

```
#define Tone_A1  262       // 440Hz
beep (Tone_A1, 10, 20)     // 100ms de son, 200ms de pause
beep (Tone_A1, 35, 42)     // 350ms de son, 420ms de pause
```

Exemple Basic :

```
#define Tone_A1  262       ' 440Hz
beep (Tone_A1, 10, 20)     ' 100ms de son, 200ms de pause
beep (Tone_A1, 35, 42)     ' 350ms de son, 420ms de pause
```

Au lieu d'utiliser la fonction beep et beep_t, vous pouvez aussi utiliser les macros.

```
sound(pitch)

und

sound_off()
```

Vous pouvez uniquement régler la hauteur de son. Cela est très utile pour une suite de sons continue (par exemple une sirène d'alarme). Cette fonction n'est PAS bloquante ! Seule la hauteur de son sera réglée et enchaînera directement avec le programme. Vous pouvez arrêter le générateur de son grâce à sound_off().

Vous avez une autre alternative possible avec le macro suivant.

```
tone(PITCH, TIME)
```

Il est à utiliser, par exemple, dans un programme d'exemple pour jouer de longues mélodies, pendant lesquelles le beeper ne doit surtout pas être éteint.

Attention : pour toutes les fonctions de beeper, la zone autorisée de « pitch » (hauteur de son » se trouve entre 0 et 65535. 0 étant la plus haute fréquence et 65535 la plus basse !

Référez-vous à la documentation du C-Control Pro, au chapitre sur le minuteur pour plus de détails. Timer3 avec Prescaler 64 est utilisé pour les fonctions beep.

3.6. Bus I²C

Vous avez en plus des fonctions bus I²C de la bibliothèque normale CCPRO, des fonctions supplémentaires spécifiques permettant de contrôler le robot depuis le programme « RP6Bases_I2CSlace ». Ce programme doit être bien évidemment charger depuis la carte mère sur un contrôleur via RP6Loader (cf. mode d'emploi RP6). Il met à disposition une rangée de registres, sur lesquels le robot est entièrement contrôlable par

bus. Vous pouvez ainsi lui envoyer des ordres et lire toutes les données des capteurs.

L'interpréteur CCPRO met à votre disposition d'autres fonctions, avec lesquelles les données sont transmises sur le bus I²C.

3.6.1. Envoyer des commandes

Un ordre seul est transmis grâce à la fonction suivante :

```
void RP6_writeCMD(byte adr, byte cmd)
Sub RP6_writeCMD(adr As Byte, cmd As Byte)
```

Généralement, il vaut mieux ne pas envoyer qu'un ordre seul, mais transmettre avec quelques valeurs – par exemple, afin de régler les LED sur la carte mère, régler également la vitesse.

Un paramètre supplément peut être transmis grâce à cette fonction :

```
void RP6_writeCMD_1param(byte adr, byte cmd, byte param)
Sub RP6_writeCMD_1param(adr As Byte, cmd As Byte, param As Byte)
```

Exemple CompactC :

```
// Adresse bus I2C du contrôleur sur la carte mère :
#define RP6_BASE_ADR 10
RP6_writeCMD_1param(RP6_BASE_ADR, CM_SETLEDS, 0x09) ;
```

Exemple Basic :

```
#define RP6_BASE_ADR 10
RP6_writeCMD_1param(RP6_BASE_ADR, CM_SETLEDS, 0x09)
```

0x09 est une hexadécimale pour le binaire : 00001001. Les 6 LED de la carte mère sont affectées aux 6 premiers bits d'un octet.

0x01 allume donc la LED1. 0x02 = LED2, 0x04 = LED3, 0x08 = LED4, 0x10 = LED 5 et 0x20 = LED 6.

On peut bien évidemment l'écrire avec des décimales : 1 pour LED1, 4 pour LED3, 16 pour LED5 et 32 pour LED6. Lorsque vous souhaitez allumer plusieurs LED, il vous faut simplement ajouter ces valeurs. Ainsi, la valeur 63 permet d'allumer toutes les LED ou 0x3F en écriture hexadécimale. 0x24 allume les LED6 et LED3.

(0x24 = 00100100 en binaire)

Il existe aussi d'autres définitions adaptées pour les LED, que vous pouvez écrire ainsi :

```
RP6_writeCMD_1param(RP6_BASE_ADR, CMT_SETLEDS, LED1 | LED4) ; // Compact C
RP6_writeCMD_1param(RP6_BASE_ADR, CMT_SETLEDS, LED1 Or LED4)  // Basic
```


Cela est déjà plus lisible. Les autres exemples en haut ne servent qu'à la compréhension, car LED1 n'est rien d'autre que la valeur décimale 1, LED2 = 2, LED3 = 4, LED4 = 8 et LED5 = 16.

Les deux commandes ci-dessus (0x09) allument donc les deux LED vertes 1 et 4.

Similaire à writeCMD_1param, il y a des fonctions pour 2 :

```
void RP6_writeCMD_2params(byte adr, byte cmd, byte param1, byte param2)
Sub RP6_writeCMD_2params(adr As Byte, cmd As Byte, param1 As Byte,
                             param2 As Byte)
```

et pour 3 paramètres :

```
void RP6_writeCMD_3params(byte adr, byte cmd, byte param1,
                             byte param2, byte param3)
Sub RP6_writeCMD_3params(adr As Byte, cmd As Byte, param1 As Byte,
                             param2 As Byte, param3 As Byte)
```

Ainsi la plupart des choses sont desservies. Pour les cas particuliers avec plus de paramètres, vous pouvez utiliser la fonction !

```
void RP6_writeCommand_params(byte adr, byte cmd,
byte params[], byte param_count)
Sub RP6_writeCommand_params(adr As Byte, cmd As Byte, ByRef params
As Byte, param_count As Byte)
```

Jusqu'à 255 paramètres peuvent être transmis à un array.

Exemple CompactC :

```
Byte params [16] ;
params [0] = 10 ;
params [1] = 44 ;
params [2] = 10 ;
// ...
params [15] = 255 ;
// Ordre « 40 » avec 16 valeurs de paramètre à envoyer à l'adresse 10 :
RP6_writeCommand_params(10, 40, params , 16) ;
```

Exemple Basic :

```
Byte params (16)
params (0) = 10
params (1) = 44
params (2) = 10
' ...
params (15) = 255 ;
' Ordre « 40 » avec 16 valeurs de paramètre à envoyer à l'adresse 10 :
RP6_writeCommand_params(10, 40, params , 16)
```